Formal Methods in Specification and Synthesis of Petri net based Reprogrammable Logic Controllers

MARIAN ADAMSKI

Computer Engineering and Electronics Institute, Technical University of Zielona Góra, 50 Podgórna Street, 65-246 Zielona Góra, POLAND, M.Adamski@iee.pz.zgora.pl

Abstract

The goal of the paper is to present a novel approach to Application Specific Logic Controllers realisations, which is suitable especially for small, embedded system designs. A discrete model of a Logic Controller is derived directly from Control Interpreted Petri Net or related Sequential Function Chart (SFC), and synthesised as a dedicated microsystem. The unified, formal intermediate model is reflected in Programmable Logic (CPLD or FPGA), by means of the direct mapping of logic conditional expressions (declarative rules) in programmable hardware. The desired behaviour of the designed logic controller can be validated by simulation in VHDL environment. The Reprogrammable (Reconfigurable) Logic Controller is implemented from VHDL textual front-end entry.

1. Introduction

The logic controller model is considered as Concurrent State Machine with Data Path, or in higher level of abstraction as Concurrent Program State Machine with Data Flow Part.

A Petri net [6,10,13] that is related with SFC (IEC 1131-3 standard) is treated as a formal model for logic rules (Gentzen sequents), as well as a suitable model for RTL-level synthesis from VHDL (Fig.1). The paper covers some effective techniques for computer-based synthesis of Reprogrammable Logic Controllers, from Petri net level to the logic design level.

In the paper it is shown how to implement parallel (concurrent) controllers in FPL (Field Programmable Logic). The main goal of the proposed design methodology is to continuously preserve the direct, self-evident correspondence among a control interpreted Petri net and its several possible hardware implementations.

The symbolic specification of Petri net, which is considered here, is given mainly in terms of local events. The local state changes are recognized and distinguished as separated transitions, with their input and output places. Another well-known possibility it is a local state oriented specification, in which conditions for token entering and holding are given, for any Petri net place (the place-based specification).

Textual specification is related with Decision Rules, which are given in the form of symbolic logic expressions.

At the beginning of design process, we use the control-oriented and Petri net based initial specification of dedicated, reactive discrete-event systems. After analysis of some behavioural and structural properties of Petri net [11], a discrete-event model is related with a knowledge-based, textual, descriptive form of representation.

The well-structured formal specification, which is represented in the human-readable language, has a direct impact on the validation, formal verification and implementation of digital microsystems in Field Programmable Logic (FPL). The declarative, logic-based specification of Petri net can increase the efficiency of the Concurrent (Parallel) Controller design [1,4,7,8,14].

In the presented view, a control automaton, with distinguished, discrete and composite states, is also treated as a dynamic inference system, based on Gentzen
logic [1]. The symbolic sequents-axioms may include some elements, taken from temporal logic [1,14]. The state space graph of a controller is considered as a compact description of a discrete transition system [10]. The behavioural statements about the functionality of the designed system (specific axioms, statements about the intended functionality of the designed discrete system) are represented by means of sequents-assertions. They form the rule-based decision system. Complex sequents are formally transformed into the set of equivalent sequent-clauses (simple sequents), which are very similar to the production rules [14]. The formally transformed decision rules are directly mapped into VHDL statements.

The Logic Controller is implemented in Field Programmable Logic, as a FPGA based reprogrammable unit. Petri net may be verified, and then transformed into an intermediate format, which is accepted by modern, VHDL-based CAD environment. The logic (Boolean) expressions, which are suitable for direct mapping into FPGA or CPLD, can be also derived and used as an input to other traditionally used tools.

The paper presents the outline of formal methodology for Application Specific Logic Controllers (ASLC) design. The previous work on that subject has been recently summarised in papers [2,3,4].

### 2. Interpreted Petri Net model of Concurrent State Machine

As an example (Fig. 1) we have selected the simplified version of Logic Controller behaviour taken from papers [2,3]. Control Interpreted Petri Net [6] has been shown to be a powerful tool to specify and model the behaviour of parallel (concurrent) controllers. The specification is given in terms of the local state changes, which are eventually supplemented by conditions for local state holding. An event driven system can be abstracted as a concurrent state machine (CSM), in which several local states (represented as places in Petri net) may change, when event occurs (transition in Petri net fires). The marking (distribution of tokens among places) of a Petri net can be regarded as the current global state of the modelled system. From the present global internal state (collection of simultaneously holding local states), the concurrent state machine goes to the next distributed internal global state, simultaneously generating the desired combinational and registered output signals \[ y \]. In such a way, an explicit local change of the marking, during the occurrence of transition, corresponds to an implicit global state change. The colours (in our example [1] and [2]), which are attached to places, distinguish and validate the consistency of the intended sequential processes (Fig. 1).

The synchronous Petri nets [5,10,12] are introduced to model binary systems, which are synchronised by a global clock. Input signals of controller are associated with transitions as a Boolean guard. The common Moore type output signals are linked with places (Fig. 1). Some Mealy type output signals might be related both with places and input signals (Fig. 2). The introducing of Mealy inputs (Fig. 1) usually makes the skeleton Petri net much simpler, but its interpretation becomes more complicated and difficult to understand (Fig 2).

A part of output changes synchronously coincides with the firing of transitions. That makes it possible to label net transitions with some names of selected, registered output signals.

To obtain the economical implementation and easy maintenance, the Petri net may be directly mapped into the Boolean equations without explicit enumeration of all possible global states and all possible global state changes. Since the specification is given only in terms of the local state changes (local transitions), the structured local state assignment (place encoding) is used [1].

### 3. Modular structured implementation of Petri net

Petri nets provide a unified method for the design of discrete-event systems from an initial system description (Fig. 1) to possibly hierarchical physical realizations.

The hierarchically structured Petri net consists of subnets, which, except possibly the Base Net (Fig. 3) are well-formed blocks. The concurrency relation between subnets is depicted by means of colours, which are
attached explicitly to the places, and implicitly to the transitions and arcs as well as to the tokens [15]. The set of subnets is partially ordered (Fig. 3, Fig. 4, Fig. 1). The coloured hierarchy relation tree (Fig.5) graphically represents the hierarchy and concurrency relations among subnets. The Base Net $MP0$ is on the root of the tree. It contains the double-macroplaces $MP1-MP7$, which stand for the hierarchically structured subnets at the lower level of hierarchy.

Each double macroplace corresponds to a compound operation, which is itself a discrete sub-process described by the doubled block. The colours $[1]$ and $[2]$ are used for distinguishing particular intended sequential processes, and continuously controlling the place invariants (P-subnets) and hierarchy tree during the composition or reduction of the net.

![Fig. 4: First and second order macronet](image)

![Fig. 5: Hierarchy tree](image)

The Petri net (Figure 1) is hierarchically encoded by means of state variables $Qi$, $i = 1,2,3,4$. The symbols $Qi$ or $/Qi$, attached to the particular path, which is directed from the root to the leave, form the unique encoding term for the considered macroplace or place.

3. Gentzen Sequent Logic

Petri nets can be viewed as a formal model for logic rule-based specification (interpretation structure). Transition rules are usually treated as production rules (‘if-then’ non-procedural statements). The principal design language used to specify the Logic Controller behaviour in extended nested If-Then-Else form in our design environment is Gentzen Sequent Logic [9].

While formulae $F$ may be regarded as a formal representation of compound proposition, sequent $|- F$ in our approach represents asserted statement. Sequents may also formally describe all general forms of conditional assertions, for example production rules: $F |- G$.

The Gentzen formal system naturally simulates and records human-like reasoning. The synthesis, based on Gentzen calculus, is treated as a formal symbolic transformation of the initial set of sequents (specification) into another equivalent set of sequents (implementation) [1,2]. The rules of inference are directly based on Gentzen Logic or they are previously proven, so the implementations are correct by construction.

4. Petri net specification in sequent logic language

The Logic Controller is considered as an abstract reasoning system (rule based system) implemented in reconfigurable hardware. The mapping between inputs, outputs and local internal states of the system is described in a formal manner by means of logic rules (represented as sequents) with some temporal operators, especially with operator 'next' @ [1,11,14]. The correctness preserving synthesis, based on Gentzen calculus, is treated as a formal transformation of the initial set of compound rules (Specification) into another set of compound rules (Implementation).

As a basic form of Petri net specification in decision rule format, the transition-oriented declarative specification is presented. It describes all possible active events in concurrent state machine, when local states associated to transition change and the guard (Boolean label) is true. The presented form of description is very closed to well-known production rules, whose are a principal forms of Petri net description in LOGICIAN [1], CONPAR [8,10], PARIS [12], and PeNCAD [3,15].

$T1$: $P1 \cdot X0 |- @P2 * @P4$;
$T2$: $P2 \cdot X1 |- @P3$;
$T3$: $P4 \cdot X3 |- @P5$;
$T4$: $P3 \cdot P5 |- @P6 * @P7$;
$T5$: $P6 \cdot X5*X6 |- @P8$;
$T6$: $P7 \cdot /X2*/X4 |- @P9$;
$T7$: $P8 \cdot /X5 |- @P6$;
$T8$: $P6 * P9 * /X6 |- @P1$

The static (level) Moore type outputs depend directly on place markings:

$P1 |- Y0$; $P2 |- Y1$; $P4 |- Y2$;
The total discrete state space (Fig. 6), which includes 9 global states, should be always consistent with all intended local state changes:

\[
\begin{align*}
&P7 \models Y3 \neq Y4; \quad P8 \models Y5; \quad P9 \models Y6. \\
\text{In some cases, like implementations with D flip-flops in FPGA, the declarative, place oriented specification is taken into account. For example, the sequents which include explicit transition symbols }\{T1, T2, ..., T8\}, \text{ after mapping the Petri net into VHDL statements in M. Bolton’s style, give economical implementations in FPGA [8].} \\
\end{align*}
\]

\[\text{Preconditions:} \]
\[P1 \neq X0 \models T1; \quad P2 \neq X1 \models T2; \quad \ldots \quad P6 \neq P9 \models X6 \models T8; \]

\[\text{Next markings:} \]
\[T8 \models P1 \models T1 \models P6 \models T8 \models P8 \models Y3 \models Y4 \models Y5 \models Y6 \models Y0 \]

\[\text{In this kind of specification, if the next value of the temporal variable, for example }\models P1, \text{ cannot be proved in the current marking (global state) as }\text{true, it is considered that it takes the value }\text{false.} \]

5. Petri Net and Logic Design

The direct mapping of a Petri net into Field Programmable Logic (FPL) is based on a self-evident correspondence between a place and a clearly defined bit-subset of a state register. The places of the Petri net are assigned to the particular flip-flops in the Register Block. VHDL supports conditional-statement constructs, which can be used to describe Petri net. The proper local state assignment (encoding) makes it possible to map a given Interpreted Petri net directly into FPGA or CPLD without its transformation into an equivalent global State Machine.

The simplest technique for Petri net place encoding is to use one-to-one mapping of places onto flip-flops in the style of a one-hot state assignment. In that case, a name of the place becomes also a name of the related flip-flop. The flip-flop is set into 1 if and only if the particular place holds the token. Some of the recent developments involving modelling and analysis such constructs in VHDL were reported, for example in [2,3,8,10,15].

In general, places after encoding are distinguished by conjunctions, which are formed from state variables from the set \{Q1, Q2, ..., Qk\}. The local states, which are active simultaneously, have non-orthogonal codes. They are represented by places holding the tokens concurrently and belonging to the same vertex from the implicitly or explicitly given reachability graph of Petri net. The local states, which belong to the different, but sometimes overlapping sequential processes \(P\)-invariants, SM-components) have orthogonal codes. One
The recommended particular method of place encoding is based on hierarchical decomposition of the net (Figure 2, Figure 3). The result of an efficient heuristic hierarchical local state assignment \([Q_1, Q_2, Q_3, Q_4]\) is as follows:

\[
\begin{align*}
P_1 &= 0 - - - \\
P_1 &= /Q_1 \\
P_2 &= 1 0 0 * \\
P_2 &= Q_1*/Q_2*/Q_3 \\
P_3 &= 1 0 1 * \\
P_3 &= Q_1*/Q_2*Q_3 \\
P_4 &= 1 0 * 0 \\
P_4 &= Q_1*/Q_2*Q_4 \\
P_5 &= 1 0 * 1 \\
P_5 &= Q_1*/Q_2*Q_4 \\
P_6 &= 1 1 0 * \\
P_6 &= Q_1*Q_2*/Q_3 \\
P_7 &= 1 1 0 * \\
P_7 &= Q_1*Q_2*Q_4 \\
P_8 &= 1 1 1 * \\
P_8 &= Q_1*Q_2*Q_3 \\
P_9 &= 1 1 * 1 \\
P_9 &= Q_1*Q_2*Q_4
\end{align*}
\]

The global state encoding is correct if all vertices of the reachability graph have different codes. The total code of the reachability graph vertex would be obtained by merging the codes of the simultaneously marked places. The code of the particular place or macroplace is represented by means of the vector composed from \{0, 1, - , *\} or it is given as a related Boolean term. The symbols 0, 1, - (‘don’t care’) have the usual meanings, but the symbol * in vector denotes ‘explicitly don’t know’ (0 or 1, but not ‘don’t care’).

For several practical applications it is recommended to manipulate with Boolean expressions (product terms), in which the symbols of places are substituted by encoding conjunctions, for example:

\[
\begin{align*}
T_1: & \quad /Q_1 * X_0 |-@Q_1*Q_2*Q_3*Q_4; \\
T_2: & \quad Q_1*/Q_2*/Q_3* X_1 |-@Q_1*Q_2*Q_3; \\
T_3: & \quad Q_1*/Q_2*/Q_4* X_3 |-@Q_1*Q_2*Q_4; \\
T_4: & \quad Q_1*Q_2*/Q_3*Q_4 |-@Q_1*Q_2*Q_3*Q_4; \\
T_5: & \quad Q_1*Q_2*/Q_3*X_5*X_6|-@Q_1*Q_2*Q_3; \\
T_6: & \quad Q_1*Q_2*/Q_4*/X_2*/X_4|-@Q_1*Q_2*Q_3*Q_4; \\
T_7: & \quad Q_1*Q_2*Q_3*/X_5|-@Q_1*Q_2*Q_3*Q_4/; \\
T_8: & \quad Q_1*Q_2*/Q_3*Q_4*/X_6|-@Q_1;
\end{align*}
\]

The simplified sequent specification, planned for implementations based on the state register with JK flip-flops, on the right sides does not contain signals, which conserve their values during the occurrences of transitions:

\[
\begin{align*}
T_1: & \quad /Q_1 * X_0 |-@Q_1*Q_2*Q_3*Q_4; \\
T_2: & \quad Q_1*/Q_2*/Q_3* X_1 |-@Q_3; \\
T_3: & \quad Q_1*/Q_2*/Q_4* X_3 |-@Q_4;
\end{align*}
\]

6. Rapid modelling and synthesis with VHDL

A particular form of place-based rules describes separately the conditions for marking and clearing all the considered places. The condition for the new marking of place P1 is described as follows:

Asserted that if P6 and P9 and not X6 then next P1;

\[\text{P6} \land \text{P9} \land \neg \text{X6} \Rightarrow @\text{P1};\]

Place P1 holds the token if it is marked and guard related with its output transition is false:

Asserted that if P1 and not X0 then next P1;

\[\neg \text{P1} \land \neg \text{X0} \Rightarrow @\text{P1};\]

After the simple formal manipulations two separated rules can be merged into the one statement

Asserted that if P6 and P9 and not X6 or P1 and not X0 then next P1;

\[\neg \text{P6} \land \neg \text{P9} \land \neg \text{X6} \lor \neg \text{P1} \land \neg \text{X0} \Rightarrow @\text{P1};\]

Taking as an example output Y0, which is active if and only if the place P1 holds the token, we obtain:

Asserted that if P1 then Y1 else not Y1;

\[\neg (\text{P1} \Rightarrow \text{Y1}) \lor (\neg \text{P1} \Rightarrow \neg \text{Y1});\]

VHDL supports conditional-statement constructs, which can be used to describe Petri net-based Concurrent State Machine implementations. As an example only a part of Petri Net specification in VHDL is presented:

```
Architecture reactor_desc of reactor is
signal P : std_logic_vector(1 to 9);
begin
  --Marking Places and Firing Transitions
  P1: process (clk, reset)
  begin
    if reset='1' then P(1)<='1';
    elsif clk'event and clk='1' then
      if (P(6)=‘1’ and P(9)=‘1’ and X6=‘0’) or P(1)=‘1’ and not(X0=‘1’) then
        P(1)<=‘1’;
        P(9)<=‘0’;
      end if;
  end if;
end process;
```

```
P2: process (clk, reset)
begin
  if reset='1' then P(1)<=‘1’;
  elsif clk'event and clk='1' then
    if (P(6)=‘1’ and P(9)=‘1’ and X6=‘0’) or P(1)=‘1’ and not(X0=‘1’) then
      P(1)<=‘1’;
      P(9)<=‘0’;
    end if;
  end if;
```

```
end process;
```

```
T1: Q1*/Q2*Q3*Q4 |-@Q2*Q3 *@/Q4;
T5: Q1*Q2*/Q3*X5*X6 |-@Q3;
T6: Q1*Q2*/Q4*/X2*/X4 |-@Q3;
T7: Q1*Q2*Q3*/X5 |-@Q3;
T8: Q1*Q2*/Q3*Q4*/X6 |-@Q1;
```

For Field Programmable Logic with JK flip flops, symbols @Qi can be replaced by J_Qi and symbols @/Qi respectively by K_Qi.
Formal logic language, which is complementary with Petri nets, is suitable in specifying system level designs of logic controllers, implemented in FPL. Simulating of Petri net model and its hardware implementation can be simplified by translating of rule-based description to VHDL. The simulation results, at circuit level and algorithmic level, can be compared immediately. To simulate the pair consisting of the controller and discrete object under control, the test bench must include, in addition to the Reprogrammable Controller description, a second VHDL program, which model the controlled subsystem behaviour. The next design step concentrates on the automatic synthesis of Reprogrammable Logic Controllers from their VHDL descriptions. The paper presents the hierarchical Petri net approach for synthesis, in which the modular net is mapped into the Field Programmable logic as structured, but a flat netlist. The hierarchy levels are conserved and related with some particular local state variable subsets, and clearly distinguished by the encoding vectors (encoding terms).

A concise, understandable specification can be easily locally modified. The experimental Petri net to VHDL translator has been implemented on the top of standard VHDL design tools, like ALDEC Active-HDL.

6. Conclusions

References


begin
  if reset='1' then Y0<=0'; end if;
  if (P(1)=1' and P(5)=1') or (P(9)=1' and X6=1')
    then Y6<=1';
    else Y6<=0';
  end if;
  end process;

end reactor_desc;